Using Timers in Unity

Part 4: On Screen Timer and Die Fixes

Overview

This is part four of the board making in Unity tutorials and will build off the project created in tutorial three. In part one, we created a rolling six sided die, and in part two we added some scripts to read the value of the die and display on screen, and in part three we added a responsive board to our scene. If you haven't looked at part one, two, and three, you should probably go back and do that now. In this fourth part we will add a timer to our scene and fix some of the dice functionality using another timer.

This tutorial was made for Unity 3.0 and assumes that the user has at least watched the first few Unity tutorial videos found here: http://unity3d.com/support/documentation/video/ This tutorial starts with the Unity project created during tutorial three.

This tutorial attempts to be more than a list of steps to follow, I tried to record some of the reasons for why I make choices. If you don't want to follow my logic and just want steps, look at the stuff in bold. At the end of each section is a list of the new Game Objects, Components or Functions covered.

We left off the last tutorial with a roll-able die that displayed its value on the screen and a game piece that moved around a simple board according to the roll. For this tutorial, we will add an on screen timer and fix the tendency for our die to register numbers as it is rolling across the screen.

Making an on-screen timer

To make an onscreen timer, we will use a text mesh as the basic game object, and then extend the behavior of that object with a script. Make a new 3DText GameObject and transform it to one corner of screen so that it can be seen when the game is played. Rename it to TimerText.

Since we don't want "Hello World" to be showing when our game starts, change the variable text in TextMesh component to 0.00 because we know that our timer will start a count from 0.
To keep track of time, we need to add a Javascript to the Project; name the new Javascript TrackTime. Here is the contents for the timer:

```javascript
var totalTime = 0.0;

private var textMeshComponent;

// store the text component so that GetComponent isn't called every frame
function Start() {
    textMeshComponent = GetComponent(“TextMesh”);
}

// update is called every frame
function Update() {
    // add amount of time since last frame to the totalTime
    totalTime += Time.deltaTime;

    // use substring to get a shorter numerical value
    textMeshComponent.text = totalTime.ToString().Substring(0,4);
}
```

We are accessing a new class and one of its values in this script: the class is Time, and the value is Time.deltaTime. The value Time.deltaTime is updated every frame, and registers the time that has passed since the last time the frame ran. This is very useful because the frame rate of our application may be variable, and knowing how much time has passed since the last frame will allow action to happen independent of frame rate.

Another new function is String.Substring() which returns just a fragment of a string, or a sub-string. The first argument is the location of the first character, and the second argument is the location of the last character. In our case, the string will be limited to just four pieces, like 0.00, or 10.1 or 500. ．
In the Update() function we just look at the amount of time that has passed, and add that value to the
totalTime. Then we access the text value in the TextMesh Component, get a string from the numerical
totalTime value (remember, strings and numbers don't mix nicely), and pull the first few characters from
that string.

If everything is working correctly, the current time should be displayed on the screen by the 3DText.

Functions Used in this Section:


String.Substring(beginIndex, endIndex) - not documented on Unity website. Takes as arguments the
beginning and the end indexes of the desired substring, and returns that substring.

Fix die false number trigger problem with deltaTime

We've had a chance now to test the die we've been working on since Tutorial 1, and it isn't perfect. One
major problem is the tendency for the die to register values as it continues to roll across our playing
space. This is caused by the triggers hitting the GroundPlane collider as it tumbles, so a useful feature
would be for the die to only update if it lays on a side for one second or more.

This is a perfect opportunity to put the Time.deltaTime value to work. We can alter the SideTrigger script
to check the amount of time on one side before updating die value:

public var faceValue = 0;

public var idleTime = 1.0; // amount of time to sit idle before setting value

private var dieValueComponent; // cache the dieValueComponent
private var totalTimeTriggered = 0.0;

function Start() {
    dieGameObject = GameObject.Find("SixSidedDie");
    dieValueComponent = dieGameObject.GetComponent("DieValue");
}

// called when colliders first meet
function OnTriggerEnter( other : Collider ) {
    // make sure that totalTimeTriggered is zero
    totalTimeTriggered = 0.0;
    Debug.Log(faceValue + " entered " + other.name);
}

// called while colliders are touching
function OnTriggerStay( other : Collider ) {
    // add to total time
    totalTimeTriggered += Time.deltaTime;

    // only change the die value if die has been trigger for idleTime seconds
    if (totalTimeTriggered > idleTime) {
        dieValueComponent.currentValue = faceValue;
    }
}

// called when collider leaves
function OnTriggerExit( other : Collider ) {
    // clear the total time

totalTimeTriggered = 0.0;

This script just uses Time.deltaTime to keep track of the time it has spent laying on one face. The total time is reset each time the trigger is entered or exited so that there is an accurate running total. The public variable idleTime allows us to alter the trigger time in the Unity Inspector tab.

If you test the scene, you will notice that the die will trigger an enter event, an exit event, but the timer won't be update. This has to do with the way we placed the triggers initially. There is a small gap between the collider that sits on the ground plane and the position of the cube we used as the trigger. This causes the trigger to fall down below the ground plane and deactivate the trigger. To fix this, just move the triggers on each side of the die to sit halfway through each face rather than floating away from it. That should get the die working properly.

End

At this point, some of the problems with the die should be fixed, and there should be an onscreen timer. This will lay the foundation for our next tutorial in which we will make a small race-against-the-clock type game.

initial part 4 - 1/24/2011 - Jonathan Cecil . jonathancecil@ucla.edu